The Care and Usage of Vectors

Tom Kelliher, CS 320

Feb. 25, 2013

1 Administrivia

Announcements

Project 1 will be due Mar. 8.

Assignment

For Friday, carefully read `collision.cpp`.

From Last Time

Discussion of Project 1.

Outline

1. Vectors. Data structures and code.
2. Discussion of `collision.cpp`.

Coming Up

Project 1 day.

2 Vectors

Operating systems is the application of data structures and algorithms. Computer graphics is the application of data structures, algorithms, trigonometry, linear algebra, Calculus, and differential equations.

1. Scalars.
2. Correspondence between points and vectors.
3. Forming a vector from two points.
4. Vector length: $|u|$.
5. Scaling vectors: αu.
6. Adding vectors.
7. Forming a new point from a point and a vector.
8. Normalizing a vector: $\hat{u} = \frac{1}{|u|}u$.

   Not the same as a normal vector.
9. Dot product properties:

   (a) Algebraic definition:
   $$u \cdot v = \sum_{i=1}^{n} u_i v_i$$

   (b) Geometric definition: $u \cdot v = \cos\theta |u||v|$.

   (c) $|u|^2 = u \cdot u$.

   ($\theta = 0$, so $\cos\theta = 1$)

   (d) $\cos\theta = \frac{u \cdot v}{|u||v|}$, where θ is the angle between u and v.

   (e) The projection of u onto v is $|u|\cos\theta = \frac{u \cdot v}{|v|}$

   Note that this is a scalar. What if I need a vector?

2.1 Data Structures and Code

This is all 2-D.

1. Basic data structure:

```
struct vec2 {

    GLfloat  x;
    GLfloat  y;

    //
    //  --- Constructors and Destructors ---
    //

    vec2( GLfloat s = GLfloat(0.0) ) :
x(s), y(s) {}

    vec2( GLfloat x, GLfloat y ) :
x(x), y(y) {}

    vec2( const vec2& v )
{ x = v.x;  y = v.y;  }

    //
    //  --- Indexing Operator ---
    //

    GLfloat& operator [] ( int i ) { return *(&x + i); }
    const GLfloat operator [] ( int i ) const { return *(&x + i); }

    //
    //  --- (non-modifying) Arithematic Operators ---
    //

    vec2 operator - () const // unary minus operator
{ return vec2( -x, -y ); }

    vec2 operator + ( const vec2& v ) const
{ return vec2( x + v.x, y + v.y ); }

    vec2 operator - ( const vec2& v ) const
{ return vec2( x - v.x, y - v.y ); }
```

```
    vec2 operator * ( const GLfloat s ) const
{ return vec2( s*x, s*y ); }

    vec2 operator * ( const vec2& v ) const
{ return vec2( x*v.x, y*v.y ); }

    friend vec2 operator * ( const GLfloat s, const vec2& v )
{ return v * s; }

    vec2 operator / ( const GLfloat s ) const {
#ifdef DEBUG
if ( std::fabs(s) < DivideByZeroTolerance ) {
    std::cerr << "[" << __FILE__ << ":" << __LINE__ << "] "
      << "Division by zero" << std::endl;
    return vec2();
}
#endif // DEBUG

GLfloat r = GLfloat(1.0) / s;
return *this * r;
    }

    //
    //  --- (modifying) Arithematic Operators ---
    //

    vec2& operator += ( const vec2& v )
{ x += v.x;  y += v.y;   return *this; }

    vec2& operator -= ( const vec2& v )
{ x -= v.x;  y -= v.y;  return *this; }

    vec2& operator *= ( const GLfloat s )
{ x *= s;  y *= s;   return *this; }

    vec2& operator *= ( const vec2& v )
{ x *= v.x;  y *= v.y; return *this; }

    vec2& operator /= ( const GLfloat s ) {
#ifdef DEBUG
if ( std::fabs(s) < DivideByZeroTolerance ) {
    std::cerr << "[" << __FILE__ << ":" << __LINE__ << "] "
      << "Division by zero" << std::endl;
}
```

```
#endif // DEBUG

GLfloat r = GLfloat(1.0) / s;
*this *= r;

return *this;
    }

    //
    //  --- Insertion and Extraction Operators ---
    //

    friend std::ostream& operator << ( std::ostream& os, const vec2& v ) {
return os << "( " << v.x << ", " << v.y <<  " )";
    }

    friend std::istream& operator >> ( std::istream& is, vec2& v )
{ return is >> v.x >> v.y ; }

    //
    //  --- Conversion Operators ---
    //

    operator const GLfloat* () const
{ return static_cast<const GLfloat*>( &x ); }

    operator GLfloat* ()
{ return static_cast<GLfloat*>( &x ); }
};
```

2. Vector length:

```
GLfloat distanceSquared(vec2 v)
{
   return dot(v, v);
}


inline GLfloat length( const vec2& v ) {
    return std::sqrt( dot(v,v) );
}
```

Square root and division are expensive. Avoid where possible.

3. Scalar, vector product:

```
GLfloat s = 5.0;
vec2 x, y(1.0, 2.0);
x = s * y;
```

4. Dot product:

```
inline GLfloat dot( const vec2& u, const vec2& v ) {
    return u.x * v.x + u.y * v.y;
}
```

5. Vector normalization:

```
inline vec2 normalize( const vec2& v ) {
    return v / length(v);
}
```

6. Vector sum:

```
vec2 x, y(1.0, 2.0);
vec2 z(y);
x = y + z;
```

3 collision.cpp

Start reading at `main()`.